Optimisation of Constant Matrix Multiplication Operation Hardware Using a Genetic Algorithm

Andrew Kinane, Valentin Muresan, and Noel O'Connor

Centre for Digital Video Processing, Dublin City University, Dublin 9, Ireland
kinanea@eeng.dcu.ie

Abstract. The efficient design of multiplierless implementations of constant matrix multipliers is challenged by the huge solution search spaces even for small scale problems. Previous approaches tend to use hill-climbing algorithms risking sub-optimal results. The three-stage algorithm proposed in this paper partitions the global constant matrix multiplier into its constituent dot products, and all possible solutions are derived for each dot product in the first two stages. The third stage leverages the effective search capability of genetic programming to search for global solutions created by combining dot product partial solutions. A bonus feature of the algorithm is that the modelling is amenable to hardware acceleration. Another bonus feature is a search space reduction early exit mechanism, made possible by the way the algorithm is modelled. Results show an improvement on state of the art algorithms with future potential for even greater savings.

1 Introduction

Applications involving the multiplication of variable data by constant values are prevalent throughout signal processing. Some common tasks that involve these operations are Finite Impulse Response filters (FIRs), the Discrete Fourier Transform (DFT) and the Discrete Cosine Transform (DCT). Optimisation of these kinds of constant multiplications will significantly impact the performance of such tasks and the global system that uses them. The examples listed are instances of a more generalised problem – that of a linear transform involving a constant matrix multiplication (CMM). The problem is summarised as follows: substitute all multiplications by constants with a minimum number of shifts and additions/subtractions (we refer to both as 'additions') [1]. The optimisation criterion may be extended beyond adder count to include factors like routability, glitching etc. but is restricted to adder count in this paper.

2 Problem Statement

A CMM equation $\mathbf{y} = \mathbf{A}\mathbf{x}$ (where $\mathbf{y}, \mathbf{x}$ are N-point 1D data vectors and $\mathbf{A}$ is an $N \times N$ matrix of M-bit fixed-point constants) may be thought of as a collection of N dot products with each dot product y_i expressed as follows:

$$y_i = \sum_{j=0}^{N-1} a_{ij} x_j, \quad i = 0, \ldots, N-1. \tag{1}$$

Each constant may be represented in signed digit (SD) form:

$$a_{ij} = \sum_{k=0}^{M-1} b_{ijk} 2^k, \quad b_{ijk} \in \left\{\overline{1}, 0, 1\right\}, \quad \overline{1} \equiv -1. \tag{2}$$

Combining Eqns. 1 and 2 yields a multiplierless dot product implementation requiring only adders and shifters:

$$y_i = \sum_{j=0}^{N-1} \sum_{k=0}^{M-1} b_{ijk} 2^k x_j, \quad i = 0, \ldots, N-1. \tag{3}$$

The goal is to find the optimal sub-expressions across all N dot products in Eqn. 3 that require fewest adder resources. As reviewed below, three properties can be used in the classification of approaches to this problem: SD permutation, pattern search strategy and problem subdivision.

SD Permutation Consider that each of the $N \times N$ M-bit fixed point constants a_{ij} have a finite set of possible SD representations. For example with $M = 4$ the constant $(-3)_{10}$ can be represented as either $(00\overline{1}\overline{1})_2, (0\overline{1}01)_2, (\overline{1}101)_2, (0\overline{1}1\overline{1})_2$ or $(\overline{1}11\overline{1})_2$. To find the optimal number of adders, all SD representations of a_{ij} should be considered since for a CMM problem Canonic Signed Digit (CSD) representation is not guaranteed to be optimal (as shown in Section 5). The difficulty is that the solution space is very large [2], hence SD permutation has thus far been applied only to simpler problems [2, 3]. Potkonjak et. al. acknowledge the potential of SD permutation but choose a single SD representation for each a_{ij} using a greedy heuristic. Neither of the recent CMM-specific algorithms in the literature apply SD permutation [4, 5], but the algorithm proposed in this paper does apply it.

Pattern Search The goal of pattern searching is to find the sub-expressions in the 3D bit matrix b_{ijk} resulting in fewest adders. Usually b_{ijk} is divided into N 2D slices along the i plane (i.e. taking each CMM dot product in isolation). Patterns are searched for in the 2D slices independently before combining the results for 3D. An example 2D slice is shown in Eqn. 4, a 4-point dot product with random 8-bit SD constants.

$$y_i = \underbrace{\begin{bmatrix} 0.9375 \\ 0.921875 \\ 0.6013625 \\ 0.1328125 \end{bmatrix}^T}_{a_{ij} \text{ at } \mathbf{A} \text{ row } i} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2^{-7} \\ 2^{-6} \\ 2^{-5} \\ 2^{-4} \\ 2^{-3} \\ 2^{-2} \\ 2^{-1} \\ 2^{0} \end{bmatrix}^T \underbrace{\begin{bmatrix} 0 & 0 & 1 & \overline{1} \\ 0 & \overline{1} & 0 & 0 \\ 0 & 0 & \overline{1} & 1 \\ \overline{1} & 1 & 0 & 0 \\ 0 & 1 & 1 & \overline{1} \\ 0 & \overline{1} & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}}_{\text{2D slice of } b_{ijk}} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{4}$$

Algorithms may search for horizontal/vertical patterns (P1D) or diagonal patterns (P2D) in the 2D slice. The P1D strategy implies a two-layer architecture of a network of adders (with no shifting of addends) to generate distributed weights for each row followed by a fast partial product summation tree (PPST) to carry out the shift accumulate (Fig. 1). The P2D strategy implies a one-layer architecture (Fig. 2) of a network of adders that in general may have shifted addends (essentially merging the two layers of the P1D strategy).

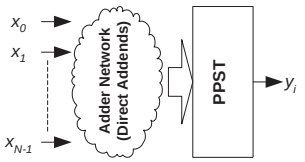

Fig. 1. P1D Architecture

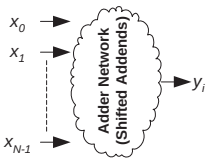

Fig. 2. P2D Architecture

Potkonjak et. al. use the P1D strategy and search for horizontal patterns while others use the P2D strategy [4, 5]. However, these approaches select sub-expressions iteratively based on some heuristic criteria that may preclude an optimal realisation of the global problem. This is because the order of sub-expression elimination affects the results [6]. The proposed algorithm sidesteps this issue by building parallel solutions using the P1D strategy.

Problem Sub-Division As in any hardware optimisation problem, synthesis issues should be considered when choosing sub-expressions for an N-point dot product (a 2D slice). If N is large (e.g. 1024-point FFT) then poor layout regularity may result from complex wiring of sub-expressions from taps large distances apart in the data vector. Indeed a recent paper has shown that choosing such sub-expressions can result in a speed reduction and greater power consumption [7]. It is therefore sensible to divide each N-point dot product into N/r sub r-point dot product chunks, where $r < N$ and $r \in \mathbb{Z}$, and optimise each chunk independently. The CMM problem hence becomes N/r independent sub problems, each with N dot products of length r (Fig. 3). The optimal choice of r is problem dependent, but the proposed algorithm currently uses $r = 4$ for reasons outlined subsequently. Eqn. 4 is an example of a sub dot product with $r = 4$.

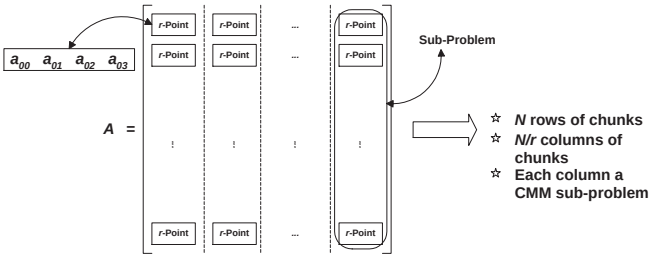

Fig. 3. CMML Divide and Conquer

3 Proposed Efficient Modelling Solution

The CMM problem is a difficult discrete combinatorial problem and currently requires a shift to a higher class of algorithms for more robust near-optimal solutions. This is because the current approaches are greedy hill-climbing algorithms and the associated results are very problem dependent [6]. The challenge is in the modelling of the problem to make it amenable to efficient computation. The algorithm proposed here models the problem in such a way as to make it amenable to so-called near-optimal algorithms (genetic algorithms (GAs), simulated annealing, tabu-search) and also hardware acceleration. The proposed approach incorporates SD permutation of the matrix constants and avoids hill-climbing by evaluating parallel solutions for each permutation. Such an approach is computationally demanding but the algorithm has been modelled with this in mind and incorporates innovative fast search techniques to reduce this burden.

The proposed algorithm permutes the SD representations of the constants in $\mathbf{A}$. For each permutation, parallel solution options are built based on different sub-expression choices. These parallel implementations are expressed as a sum of products (SOP), where each product term in the SOP represents a particular solution (with an associated adder count). The SD permutation is done on each CMM dot product in isolation (Section 4.1), and the results are subsequently combined (Section 4.2). The algorithm searches for the combined SOP that represents the overall best (in terms of adder count) sub-expression configuration to implement the CMM equation. Previous approaches derive one implementation option (akin to a single term SOP) whereas the proposed approach derives parallel implementations (a multi-term SOP). It is this multi-term SOP approach and its manipulation (Section 4) that make the algorithm suitable for GAs and hardware acceleration.

The proposed algorithm currently uses the P1D strategy, so it searches for horizontal sub-expression patterns of $\{\pm 1\}$ digits in a 2D slice. The proposed SOP modelling idea can be extended to cover the P2D strategy by simply extending the digit set from $\{\pm 1\}$ to $\left\{\pm 1, \pm 2, \pm \frac{1}{2}, \pm 4, \pm \frac{1}{4}, \ldots\right\}$. To save space, the reasoning for this idea is not elaborated upon in this paper, but is targeted as future work.

4 The Proposed CMM Optimisation Algorithm

The proposed approach is a three stage algorithm as depicted in Fig. 4. Firstly all SD representations of the M-bit fixed point constants are evaluated using an M-bit radix-2 SD counter (digit set $\left\{\overline{1}, 0, 1\right\}$). Then, each dot product in the CMM is processed independently by the dot product level (DPL) algorithm. Finally the DPL results are merged by the CMM level (CMML) algorithm. The three steps may execute in a pipelined manner with dynamic feedback between stages. This offers search space reduction potential as outlined subsequently.

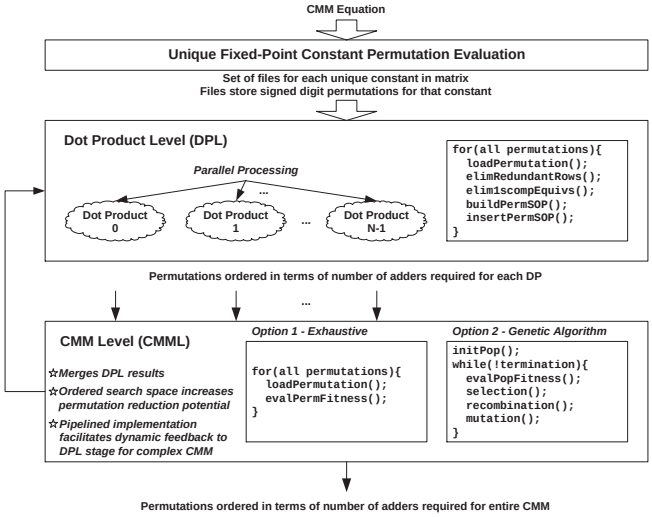

Fig. 4. Summary of the CMM Optimisation Algorithm

4.1 Dot Product Level (DPL) Stage

The DPL algorithm iteratively builds a SOP, and the final SOP terms are the unique sub-expression selection options after considering all SD permutations of the dot product constants in question. The final SOP terms are listed in increasing order of the number of adders required by the underlying sub-expressions.

Each SOP term is represented internally as a data structure with elements `p_vec` (a bit vector where each set bit represents a specific adder to be resource allocated) and `hw` (the Hamming weight of `p_vec` that records the total adder requirement). The number of possible two input additions is equivalent to the combinatorial problem of leaf-labelled complete rooted binary trees [8]. With $r = 4$, the number of possibilities is 180 (proof omitted to save space) and the general series in r increases quickly for $r > 4$. We are currently researching an automated method for configuring the DPL algorithm for any r. Currently, however, each `p_vec` is a 180-bit vector with a `hw` equal to the number of required adders.

The DPL algorithm executes for each SD permutation of the dot product constants in question, and builds a 'permutation SOP' at each iteration. This process is described in detail in [9]. The permutation SOP for Eqn. 4 is given by Eqn. 5 where p_v means bit v is set in the 180-bit `p_vec` for that SOP term.

$$\begin{aligned} &((p_{11})(p_6)(p_3)(p_{51})(p_{10})(p_0)) \text{ OR} \\ &((p_{11})(p_6)(p_{10})(p_{52})(p_0)) \text{ OR} \\ &((p_{11})(p_6)(p_{53})(p_{10})(p_0)) \end{aligned} \tag{5}$$

The first term in Eqn. 5 has `hw` = 6 so it requires 6 unique additions (+PPST) to implement Eqn. 4 whereas the latter two options only require 5 unique additions (+PPST). Obviously one of the latter two options is more efficient if implementing this dot product in isolation. However, when targeting a CMM

problem one must consider the CMM level, and it may be that permuting the first option at CMML gives a better overall result since it may overlap better with requirements for the other dot products. Hence it is necessary to store the entire SOP for each permutation at DPL and then permute these at CMML to get the guaranteed optimal.

The algorithm checks each term in the permutation SOP produced at each DPL iteration to see if it has already been found with a previous permutation. If so it is discarded – only unique implementations are added to the global list. This global list is implemented using a 2D skip list to minimise the overhead of searching it with a new term from the current permutation SOP (Fig. 5) [9]. In the horizontal direction there are 'skip nodes' ordered from left to right in order of increasing `hw` in the skip node list (SNL). In the vertical direction there are 'product nodes' and each skip node points to a product node list (PNL) of ordered product nodes where each product node in the PNL has the same number of bits set (i.e. `hw`) in its `p_vec` bit vector. When inserting a new term into the list, a unique permutation ID (`pid`) is added to the node along with `p_vec` so that the SD permutation that generated it can be reconstructed.

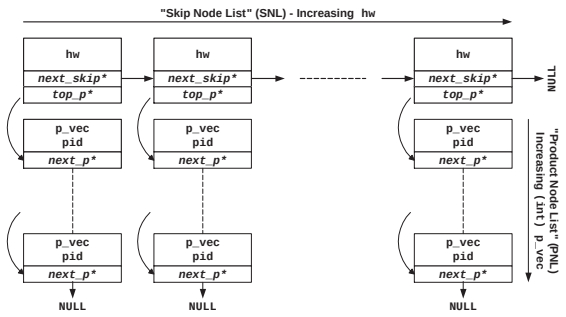

Fig. 5. DPL Skip List Arrangement

The DPL algorithm is dominated by low level operations such as comparisons, Boolean logic and bit counting. Indeed profiling shows that on average 60% of the computation time is consumed by bit counting (50%) and bitwise OR (10%). Such tasks can readily be accelerated in hardware by mapping the multi-term SOP to a FIFO structure and the logic OR operations to OR gates.

4.2 Constant Matrix Multiplication Level (CMML) Stage

Once the DPL algorithm has run for each of the dot products in the CMM, there will be N 2D skip lists – one for each of the N dot products examined. The task now is to find the best set of overlapping product nodes for all of the CMM dot products, with one node for each dot product. Overlapping nodes have similar `p_vec` set bits, and this results in adder resource sharing when implementing the CMM. It is expected (though not guaranteed) that since the skip lists are ordered with the lowest `hw` PNL first, the optimal result will be converged upon

quickly saving needless searching of large areas of the permutation space. The CMML algorithm searches for the optimal overlapping nodes from each of the DPL lists.

Exhaustive Approach An exhaustive CMML algorithm permutes the terms in each skip list with terms from others, starting from the top of each. For each permutation, N product nodes (one from each list) are combined using bitwise OR and bit counting similar to the techniques used in the DPL algorithm. The value of `hw` of the combined node represents the number of adders necessary to implement the CMM for the current permutation. The potential exists to use the lowest `hw` value found thus far to rule out areas of the search space – hence the early exit mechanism referred to previously. For example if an improved value of `hw` $= 5$ is found for a CMML solution, there is no point in searching DPL PNLs with `hw` > 5 since they are guaranteed not to overlap with other DPL PNLs and give a better result than 5. The current best value of `hw` at CMML level could also be fed back to the DPL algorithm to reduce the size of the skip lists generated by DPL (and hence permutation space) without compromising optimality. However, despite the DPL skip list ordering, the huge permutation space means that the exhaustive CMML approach is not tractable, especially as N increases.

Genetic Programming Approach The proposed modelling of the CMM problem and bit vector representation of candidate solutions means that the CMML algorithm is very amenable to GAs. The bit vectors can be interpreted as chromosomes and the value of `hw` can be used to build an empirical fitness function (the less adders required the fitter the candidate). A proposed GA to implement the CMML algorithm is summarised in Algorithm 1.

Algorithm 1: GA-based CMML Algorithm

```
init_pop();
while !termination condition do
   eval_pop_fitness();
   selection();
   recombination();
   mutation();
end
```

A candidate solution `c` is represented by a set of N pointers `slp[i][c]`, where each pointer addresses a product node in dot product skip list `i` ($i = 0, 1, \ldots, N-1$). The N product nodes are combined using bitwise OR and bit counting as described in [9]. The task of the GA is to find the DPL component product nodes that overlap as much as possible resulting in the fewest adders necessary to implement the CMM with a P1D architecture (Fig. 1). The individual steps of Algorithm 1 are described in the following sections.

Step 0 – Initialise Population The size of the population is determined by the parameter *pop_size*. Since the DPL stage results are ordered as described

in Section 4.1, the population is initialised with candidates (sets of pointers) near the top of the DPL lists. This is achieved by weighting the selection of the initial candidates. Let z represent the address each of the N component pointers `slp[i][c]` can assume for any candidate `c`. For each pointer, z is in the range $0 \leq z \leq \mathrm{NP}_i$, where NP_i is the number of product nodes in skip list `i`. The algorithm randomly sets the pointer address z for all N pointers for each of the initial *pop_size* candidates according to an exponential probability mass function Eqn. 6.

$$p(z) = \frac{1}{\mu} \exp(-z/\mu) \tag{6}$$

According to Eqn. 6, the lower the value of parameter μ, the more likely a candidate is to have DPL component pointers nearer the top of the respective DPL skip lists (i.e. z tends to zero for each of the N pointers).

Step 1 – Population Fitness Evaluation The fitness of a candidate solution is obtained by doing a bitwise OR of all of the component pointees followed by bit counting. The lower the resultant bit count the better, as it means less adder resources are required to implement the CMM problem with a P1D hardware architecture. In future work we intend extending the fitness function to include factors like fanout and logic depth, e.g. Eqn. 7. Currently Eqn. 7 is restricted to adder count only.

$$f = \alpha(\text{Adder Count}) + \beta(\text{Fanout}) + \gamma(\text{Logic Depth}) + \ldots \tag{7}$$

Step 2 – Selection A good selection method should maintain an appropriate balance between selective pressure and population diversity. The proposed method is a variation of Goldberg's Boltzmann Tournament Selection algorithm [10]. Tournament selection involves a pure random selection of t individuals ($t \leq pop\_size$) that compete in terms of fitness against each other and the winner is selected. This process is repeated *pop_size* times. However, we propose to use a strategy with a 'fuzzy' selection decision with $t = 2$. Goldberg's algorithm is based upon simulated annealing, i.e. at high 'temperatures' there is a greater chance that weak candidates may be selected, which enhances population diversity and makes it less likely that the algorithm will get stuck in local optima. As the temperature cools, the strong candidates begin to dominate selection since the algorithm should be converging on the true optimum.

The proposed approach uses Eqn. 8 which is plotted along with the exponent of $X = \frac{f(j)-f(k)}{T}$ in Fig. 6 where $f(j)$ and $f(k)$ are the fitness values of candidates j and k respectively.

$$W = \frac{1}{1 + e^{\frac{f(j)-f(k)}{T}}} = \frac{1}{1 + e^{X}} \tag{8}$$

As is clear from Fig. 6, as the temperature T decreases, the value of the exponential term X moves further from the central vertical axis for a fixed $f(j)$ and $f(k)$. As T decreases $W \to 1$ when $f(j) < f(k)$ and $W \to 0$ when $f(k) < f(j)$.

The original Boltzmann tournament selection algorithm proposed by Goldberg uses $t = 3$, and lets W equal the probability that j wins the tournament and

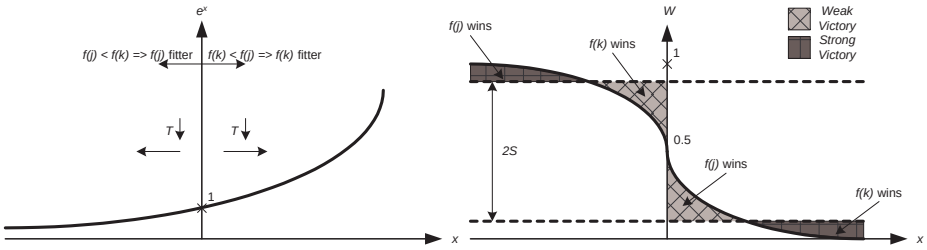

Fig. 6. Boltzmann Decision Based Simulated Annealing

$(1-W)$ be the probability that k wins the tournament [10]. We propose a variation on Goldberg's algorithm by introducing a fuzzy select threshold S to enhance the population diversity. Using S, the selection algorithm can be programmed to have a higher probability of selecting a weak candidate as a tournament victor when the temperature T is high in the early generations. As the temperature decreases and the algorithm converges on the optimum, the stronger candidate has a greater chance of victory. The approach is summarised in Algorithm 2.

Algorithm 2: Fuzzy Boltzmann Tournament Selection Algorithm

```
if f(j) < f(k) then
    if W > (0.5 + S) then j wins (strong victory);
    else k Wins (weak victory)
end
else if f(j) > f(k) then
    if W < (0.5 - S) then k wins (strong victory);
    else j Wins (weak victory)
end
else
    Choose pure random winner
end
```

To summarise, the proposed selection method maintains a balance between population diversity and selection strength. The selection decision depends on the relative fitness of competing individuals, the temperature T and the fuzzy select threshold S. Since the GA should converge on globally optimal solutions as the generations iterate, the parameters T and S should decay over the generations to select the strong candidates with higher probability.

Step 3 – Recombination After pop_size individuals have been selected, a proportion of these are further selected for uniform crossover based on a probability p_c. Since each candidate is represented by N pointers, the uniform crossover process generates a random N-bit binary mask. Each bit location in the mask determines the mixture of genetic material from the parents each offspring is created with. Consider Fig. 7. If a bit location in the mask is '0', the corresponding

pointer component for offspring '0' is created respectively from parent '0', and the corresponding component for offspring '1' is created from parent '1'. The opposite creation process occurs if the bit is '1'.

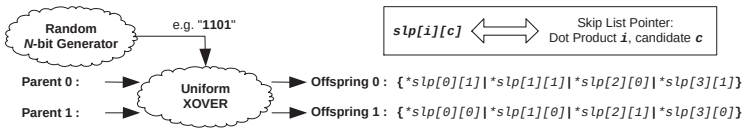

Fig. 7. Uniform Crossover Example

Step 4 – Mutation After selection and crossover, the DPL component pointers of each candidate undergo mutation based on a probability p_{mut}. If mutation is applied, the degree of mutation is determined by a value M, where $M \in \mathbb{Z}$. A pointer selected for mutation moves M pointer locations up $(M < 0)$ or down $(M > 0)$ its associated DPL skip list. The range of mutations possible depends on the value of a parameter M_{max}. The value for M is determined based on a binomial probability density function $p(M)$ Eqn. 9. This distribution means that if mutation is applied, smaller mutations are more likely than large mutations.

$$p(M) = \frac{(2M_{max} - 1)!}{M!((2M_{max} - 1) - M)!} 0.5^{M} (0.5)^{((2M_{max}-1)-M)} \tag{9}$$

To allow positive or negative mutations, the binomial distribution is re-aligned about $M = 0$ (where $p(0) = 0$ because $M = 0$ means no mutation).

After this step, the new population forming the next generation is ready and the process loops back to step 1. The process continues iterating steps 1-4 until a termination condition is met (a fixed number of generations or a time constraint).

4.3 Genetic Algorithm Parameter Selection

Choosing values for the parameters that steer a GA is a difficult problem in itself. The parameter values in Table 1 have been obtained empirically by trial and error, and future work will investigate a more sophisticated method. Based on empirical observations, the tuned parameter values in Table 1 imply that the CMML GA produces better results when there is weak selective pressure (strong diversity). The reason for this is likely to be because the variance of the solution space fitness values is quite low, according to the current fitness function, relative to the size of the solution space. Hence the current search is almost a "needle in a haystack" search, so a healthy diversity is needed. Future work on this algorithm aims to increase the dimensionality of the fitness function to include other factors like logic depth and fanout as well as adder count. Extending the fitness function should increase the granularity of the fitness values in the solution space. Hence the tuned genetic algorithm parameters are likely to change in future so that the selective pressure will increase.

Table 1. CMML Genetic Algorithm Parameters

Parameter	**Name**	**Value**
pop_size	Population Size	3000
μ	Initialisation Weight	10.0
T	Selection Temperature	0.001
S	Selection Threshold	0.4
p_c	Crossover Probability	0.98
p_{mut}	Mutation Probability	0.08
M_{max}	Max Mutation Size	6

5 Experimental Results

For a fair comparison with other approaches, the number of 1-bit full adders (FAs) allocated in each optimised architecture should be used as opposed to 'adder units', since the bitwidth for each unit is unspecified in other publications apart from in [5]. FA count more accurately represents circuit area requirements. Using the 8-point 1D DCT ($N = 8$ with various M) as a benchmarking CMM problem, Table 2 compares results with other approaches based on adder units and FAs where possible. Our approach compares favourably with [5] in terms of FAs (see FA% savings in Table 2), even though this gain is not reflected by the number of adder units required.

Our previous results were based on running the proposed CMML GA with untuned parameters for 100000 generations [9]. Using the tuned parameters of Table 1, our results clearly improve as is evident from Table 2. The tuned parameters also find these improved solutions after fewer generations (1000). For each of the benchmarks in Table 2, the tuned parameters cause the proposed algorithm to invoke its search space reduction mechanism (Section 4.2). This reduces the search space from the order of 10^{20} to 10^{17} without compromising the quality of the results , representing a reduction of more than 99%. The hypothesis of achieving extra saving by permuting the SD representations is validated by the fact that the best SD permutation corresponding to our results in Table 2 are not the CSD permutation.

Table 2. 1D 8-point DCT Adder Unit / Full Adder Requirements

CMM	Initial	[1]	[4]	[5]		Ours					
						Untuned GA [9]			Tuned GA		
	+	+	+	+	FA	+	FA	FA%	+	FA	FA%
DCT 8bit	300	94	65	56	739	78	730	1.2	77	712	3.7
DCT 12bit	368	100	76	70	1202	109	1056	12.1	108	1048	12.8
DCT 16bit	521	129	94	89	2009	150	1482	26.2	141	1290	35.8

Even given the savings illustrated in Table 2, there exists significant potential for improvement:

1. Investigation of an optimal value for r, that is the optimal sub division of large CMM problems into independent chunks. This can only be truly evaluated if synthesis parameters such as fanout and routability are included in the fitness function as well as FA count.
2. The integration of the P2D strategy mentioned earlier. It is likely that there exists an upper bound on the number of rows apart within the b_{ijk} slice between which useful sub-expressions will be found. This is because if sub-expression addends come from rows far apart in b_{ijk}, the adders inferred have a large bitwidth.
3. Extension of the fitness function as indicated, and subsequent tuning of the GA parameters.

6 Conclusions

The general multiplierless CMM design problem has a huge search space, especially if different SD representations of the matrix constants are considered. The proposed algorithm addresses this by organising the search space effectively, and by using a GA to quickly search for near optimal solutions. Experimental results validate the approach, and show an improvement on the current state of the art.

Acknowledgement

The support of the Embark Initiative and of the Informatics Commercialisation initiative of Enterprise Ireland is gratefully acknowledged.

References

1. Potkonjak, M., Srivastava, M.B., Chandrakasan, A.P.: Multiple Constant Multiplications: Efficient and Versatile Framework and Algorithms for Exploring Common Subexpression Elimination. IEEE Transactions on Computer-Aided Design of Integrated Circuits **15** (1996) 151–165
2. Dempster, A.G., Macleod, M.D.: Digital Filter Design Using Subexpression Elimination and all Signed-Digit Representations. In: Proc. IEEE International Symposium on Circuits and Systems. Volume 3. (2004) 169–172
3. Dempster, A.G., Macleod, M.D.: Using all Signed-Digit Representations to Design Single Integer Multipliers using Subexpression Elimination. In: Proc. IEEE International Symposium on Circuits and Systems. Volume 3. (2004) 165–168
4. Macleod, M.D., Dempster, A.G.: Common subexpression elimination algorithm for low-cost multiplierless implementation of matrix multipliers. IEE Electronics Letters **40** (2004) 651–652
5. Boullis, N., Tisserand, A.: Some Optimizations of Hardware Multiplication by Constant Matrices. IEEE Transactions on Computers **54** (2005) 1271–1282
6. Macleod, M.D., Dempster, A.G.: Multiplierless FIR Filter Design Algorithms. IEEE Signal Processing Letters **12** (2005) 186–189
7. Martinez-Peiro, M., Boemo, E.I., Wanhammar, L.: Design of High-Speed Multiplierless Filters Using a Nonrecursive Signed Common Subexpression Algorithm. IEEE Transations on Circuits and Systems II **49** (2002) 196–203

8. Andres, S.D.: On the number of bracket structures of n-operand operations constructed by binary operations (2005) private communication.
9. Kinane, A., Muresan, V., O'Connor, N.: Towards an Optimised VLSI Design Algorithm for the Constant Matrix Multiplication Problem. In: Proc. IEEE International Symposium on Circuits and Systems, Kos, Greece (2006)
10. Goldberg, D.: A note on Boltzmann tournament selection for genetic algorithms and population-oriented simulated annealing. Complex Systems **4** (1990) 445–460